

**APPARATUS AND METHOD FOR VIRTUALIZING A QUEUE PAIR SPACE
TO MINIMIZE TIME-WAIT IMPACTS**

BACKGROUND OF THE INVENTION

5

1. Technical Field:

The present invention is directed to an improved data processing system. More specifically, the present invention is directed to an apparatus and method for
10 virtualizing a queue pair space to minimize time-wait impacts.

2. Description of Related Art:

In a System Area Network (SAN), the hardware
15 provides a message passing mechanism that can be used for Input/Output devices (I/O) and interprocess communications (IPC) between general computing nodes. Processes executing on devices access SAN message passing hardware by posting send/receive messages to send/receive
20 work queues on a SAN channel adapter (CA). These processes also are referred to as "consumers."

The send/receive work queues (WQ) are assigned to a consumer as a queue pair (QP). The messages can be sent over five different transport types: Reliable Connected
25 (RC), Reliable datagram (RD), Unreliable Connected (UC), Unreliable Datagram (UD), and Raw Datagram (RawD). Consumers retrieve the results of these messages from a completion queue (CQ) through SAN send and receive work completion (WC) queues. The source channel adapter takes
30 care of segmenting outbound messages and sending them to the destination. The destination channel adapter takes care of reassembling inbound messages and placing them in

Docket No. AUS920010493US1

the memory space designated by the destination's consumer.

Two channel adapter types are present in nodes of the SAN fabric, a host channel adapter (HCA) and a target channel adapter (TCA). The host channel adapter is used by general purpose computing nodes to access the SAN fabric. Consumers use SAN verbs to access host channel adapter functions. The software that interprets verbs and directly accesses the channel adapter is known as the channel interface (CI).

Target channel adapters (TCA) are used by nodes that are the subject of messages sent from host channel adapters. The target channel adapters serve a similar function as that of the host channel adapters in providing the target node an access point to the SAN fabric.

The SAN channel adapter architecture explicitly provides for sending and receiving messages directly from application programs running under an operating system. No intervention by the operating system is required for an application program to post messages on send queues, post message receive buffers on receive queues, and detect completion of send or receive operations by polling of completion queues or detecting the event of an entry stored on a completion queue, e.g., via an interrupt.

When connections are established between nodes in a SAN fabric, physical queue pairs of channel adapters are typically used to facilitate the connection. When these connections are torn down, the physical queue pairs are placed in a time-wait state in order to make sure that all data packets in the SAN fabric at the time the

Docket No. AUS920010493US1

connection is torn down, have time to be routed to their destination. During this time-wait state, the physical queue pairs cannot be used to establish new connections with the same or other nodes. This results in an

5 inefficiency in the SAN architecture with regard to the establishment and tearing down of connections between nodes. Therefore, it would be beneficial to have an apparatus and method for avoiding the time-wait state delays experienced in typical SAN architectures.

T03T30"563T650

SUMMARY OF THE INVENTION

5 An apparatus and method for virtualizing a queue pair space to minimize time-wait impacts. The apparatus and method allocate virtual queue pairs from a virtual queue pair pool of a node to connections between the node and other nodes.

10 The connection is established between a physical queue pair of the node and a physical queue pair of another node. However, from the viewpoint of other nodes, they are communicating with the present node using the virtual queue pair and not the physical queue pair for the present node.

15 By using the virtual queue pairs, the same physical queue pair may be used for successive connections without the need to wait for the timewait period to elapse. Moreover, by using a virtual queue pair rather than a physical queue pair, when a connection is torn down, the virtual queue pair is placed in a time-wait state rather than the physical queue pair. As a result, the physical queue pair may continue to function while the virtual queue pair is in the time-wait state

20

US 9,343,000 B2

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a diagram of a distributed computer system is illustrated in accordance with a preferred embodiment of the present invention;

Figure 2 is a functional block diagram of a host processor node in accordance with a preferred embodiment of the present invention;

Figure 3A is a diagram of a host channel adapter in accordance with a preferred embodiment of the present invention;

Figure 3B is a diagram of a switch in accordance with a preferred embodiment of the present invention;

Figure 3C is a diagram of a router in accordance with a preferred embodiment of the present invention;

Figure 4 is a diagram illustrating processing of work requests in accordance with a preferred embodiment of the present invention;

Figure 5 is a diagram illustrating a portion of a distributed computer system in accordance with a preferred embodiment of the present invention in which a reliable connection service is used;

Figure 6 is a diagram illustrating a portion of a distributed computer system in accordance with a

Docket No. AUS920010493US1

preferred embodiment of the present invention in which reliable datagram service connections are used;

Figure 7 is an illustration of a data packet in accordance with a preferred embodiment of the present invention;

Figure 8 is a diagram illustrating a portion of a distributed computer system in accordance with a preferred embodiment of the present invention;

Figure 9 is a diagram illustrating the network addressing used in a distributed networking system in accordance with the present invention;

Figure 10 is a diagram illustrating a portion of a distributed computing system in accordance with a preferred embodiment of the present invention in which the structure of SAN fabric subnets is illustrated;

Figure 11 is a diagram of a layered communication architecture used in a preferred embodiment of the present invention;

Figure 12 is an exemplary diagram illustrating a process for creating a connection in a SAN architecture;

Figure 13 is an exemplary diagram illustrating a process for releasing or tearing down a connection in ~~is~~ a SAN architecture;

Figure 14 is an exemplary diagram illustrating a connection between two nodes in accordance with the present invention;

Figure 15 is an exemplary diagram illustrating the re-use of virtualized queue pairs before a time-wait period expires;

Figure 16 is a flowchart outlining an exemplary operation of the present invention; and

Docket No. AUS920010493US1

Figure 17 is an exemplary diagram illustrating the application of the present invention to end-to-end contexts.

FIG. 17 is a diagram illustrating the application of the present invention to end-to-end contexts.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides an apparatus and method for virtualizing a queue pair space to minimize time-wait impacts. The present invention may be implemented in hardware, software, or a combination of hardware and software. The present invention is preferably implemented in a distributed computing system, such as a system area network (SAN) having end nodes, switches, routers, and links interconnecting these components. Each end node uses send and receive queue pairs to transmit and receives messages. The end nodes segment the message into packets and transmit the packets over the links. The switches and routers interconnect the end nodes and route the packets to the appropriate end node. The end nodes reassemble the packets into a message at the destination.

Figure 1 is a diagram of a distributed computer system in accordance with a preferred embodiment of the present invention. The distributed computer system represented in **Figure 1** takes the form of a system area network (SAN) **100** and is provided merely for illustrative purposes, and the embodiments of the present invention described below can be implemented on computer systems of numerous other types and configurations. For example, computer systems implementing the present invention can range from a small server with one processor and a few input/output (I/O) adapters to massively parallel supercomputer systems with hundreds or thousands of processors and thousands of I/O adapters. Furthermore,

Docket No. AUS920010493US1

the present invention can be implemented in an infrastructure of remote computer systems connected by an Internet or intranet.

SAN **100** is a high-bandwidth, low-latency network interconnecting nodes within the distributed computer system. A node is any component attached to one or more links of a network and forming the origin and/or destination of messages within the network. In the depicted example, SAN **100** includes nodes in the form of host processor node **102**, host processor node **104**, redundant array independent disk (RAID) subsystem node **106**, and I/O chassis node **108**. The nodes illustrated in **Figure 1** are for illustrative purposes only, as SAN **100** can connect any number and any type of independent processor nodes, I/O adapter nodes, and I/O device nodes. Any one of the nodes can function as an endnode, which is herein defined to be a device that originates or finally consumes messages or frames in SAN **100**.

In one embodiment of the present invention, an error handling mechanism in distributed computer systems is present in which the error handling mechanism allows for reliable connection or reliable datagram communication between end nodes in distributed computing system, such as SAN **100**.

A message, as used herein, is an application-defined unit of data exchange, which is a primitive unit of communication between cooperating processes. A packet is one unit of data encapsulated by networking protocol headers and/or trailers. The headers generally provide control and routing information for directing the frame

Docket No. AUS920010493US1

through SAN. The trailer generally contains control and cyclic redundancy check (CRC) data for ensuring packets are not delivered with corrupted contents.

SAN **100** contains the communications and management infrastructure supporting both I/O and interprocessor communications (IPC) within a distributed computer system. The SAN **100** shown in **Figure 1** includes a switched communications fabric **116**, which allows many devices to concurrently transfer data with high-bandwidth and low latency in a secure, remotely managed environment. Endnodes can communicate over multiple ports and utilize multiple paths through the SAN fabric. The multiple ports and paths through the SAN shown in **Figure 1** can be employed for fault tolerance and increased bandwidth data transfers.

The SAN **100** in **Figure 1** includes switch **112**, switch **114**, switch **146**, and router **117**. A switch is a device that connects multiple links together and allows routing of packets from one link to another link within a subnet using a small header Destination Local Identifier (DLID) field. A router is a device that connects multiple subnets together and is capable of routing frames from one link in a first subnet to another link in a second subnet using a large header Destination Globally Unique Identifier (DGUID).

In one embodiment, a link is a full duplex channel between any two network fabric elements, such as endnodes, switches, or routers. Example suitable links include, but are not limited to, copper cables, optical cables, and printed circuit copper traces on backplanes and printed circuit boards.

Docket No. AUS920010493US1

For reliable service types, endnodes, such as host processor endnodes and I/O adapter endnodes, generate request packets and return acknowledgment packets. Switches and routers pass packets along, from the source to the destination. Except for the variant CRC trailer field, which is updated at each stage in the network, switches pass the packets along unmodified. Routers update the variant CRC trailer field and modify other fields in the header as the packet is routed.

10 In SAN **100** as illustrated in **Figure 1**, host processor node **102**, host processor node **104**, and I/O chassis **108** include at least one channel adapter (CA) to interface to SAN **100**. In one embodiment, each channel adapter is an endpoint that implements the channel adapter interface in sufficient detail to source or sink packets transmitted on SAN fabric **100**. Host processor node **102** contains channel adapters in the form of host channel adapter **118** and host channel adapter **120**. Host processor node **104** contains host channel adapter **122** and host channel adapter **124**. Host processor node **102** also includes central processing units **126-130** and a memory **132** interconnected by bus system **134**. Host processor node **104** similarly includes central processing units **136-140** and a memory **142** interconnected by a bus system **144**.

Host channel adapters **118** and **120** provide a connection to switch **112** while host channel adapters **122** and **124** provide a connection to switches **112** and **114**.

In one embodiment, a host channel adapter is implemented in hardware. In this implementation, the host channel adapter hardware offloads much of central

Docket No. AUS920010493US1

processing unit and I/O adapter communication overhead. This hardware implementation of the host channel adapter also permits multiple concurrent communications over a switched network without the traditional overhead associated with communicating protocols. In one embodiment, the host channel adapters and SAN **100** in **Figure 1** provide the I/O and interprocessor communications (IPC) consumers of the distributed computer system with zero processor-copy data transfers without involving the operating system kernel process, and employs hardware to provide reliable, fault tolerant communications.

As indicated in **Figure 1**, router **116** is coupled to wide area network (WAN) and/or local area network (LAN) connections to other hosts or other routers. The I/O chassis **108** in **Figure 1** includes an I/O switch **146** and multiple I/O modules **148-156**. In these examples, the I/O modules take the form of adapter cards. Example adapter cards illustrated in **Figure 1** include a SCSI adapter card for I/O module **148**; an adapter card to fiber channel hub and fiber channel-arbitrated loop (FC-AL) devices for I/O module **152**; an ethernet adapter card for I/O module **150**; a graphics adapter card for I/O module **154**; and a video adapter card for I/O module **156**. Any known type of adapter card can be implemented. I/O adapters also include a switch in the I/O adapter backplane to couple the adapter cards to the SAN fabric. These modules contain target channel adapters **158-166**.

In this example, RAID subsystem node **106** in **Figure 1** includes a processor **168**, a memory **170**, a target channel adapter (TCA) **172**, and multiple redundant and/or striped

Docket No. AUS920010493US1

storage disk unit **174**. Target channel adapter **172** can be a fully functional host channel adapter.

SAN **100** handles data communications for I/O and interprocessor communications. SAN **100** supports

5 high-bandwidth and scalability required for I/O and also supports the extremely low latency and low CPU overhead required for interprocessor communications. User clients can bypass the operating system kernel process and directly access network communication hardware, such as
10 host channel adapters, which enable efficient message passing protocols. SAN **100** is suited to current computing models and is a building block for new forms of I/O and computer cluster communication. Further, SAN **100** in **Figure 1** allows I/O adapter nodes to communicate among
15 themselves or communicate with any or all of the processor nodes in distributed computer system. With an I/O adapter attached to the SAN **100**, the resulting I/O adapter node has substantially the same communication capability as any host processor node in SAN **100**.

20 In one embodiment, the SAN **100** shown in **Figure 1** supports channel semantics and memory semantics. Channel semantics is sometimes referred to as send/receive or push communication operations. Channel semantics are the type of communications employed in a traditional I/O
25 channel where a source device pushes data and a destination device determines a final destination of the data. In channel semantics, the packet transmitted from a source process specifies a destination processes' communication port, but does not specify where in the
30 destination processes' memory space the packet will be

Docket No. AUS920010493US1

written. Thus, in channel semantics, the destination process pre-allocates where to place the transmitted data.

In memory semantics, a source process directly reads or writes the virtual address space of a remote node destination process. The remote destination process need only communicate the location of a buffer for data, and does not need to be involved in the transfer of any data. Thus, in memory semantics, a source process sends a data packet containing the destination buffer memory address of the destination process. In memory semantics, the destination process previously grants permission for the source process to access its memory.

Channel semantics and memory semantics are typically both necessary for I/O and interprocessor communications. A typical I/O operation employs a combination of channel and memory semantics. In an illustrative example I/O operation of the distributed computer system shown in **Figure 1**, a host processor node, such as host processor node **102**, initiates an I/O operation by using channel semantics to send a disk write command to a disk I/O adapter, such as RAID subsystem target channel adapter (TCA) **172**. The disk I/O adapter examines the command and uses memory semantics to read the data buffer directly from the memory space of the host processor node. After the data buffer is read, the disk I/O adapter employs channel semantics to push an I/O completion message back to the host processor node.

In one exemplary embodiment, the distributed computer system shown in **Figure 1** performs operations that employ virtual addresses and virtual memory protection mechanisms to ensure correct and proper access

Docket No. AUS920010493US1

to all memory. Applications running in such a distributed computed system are not required to use physical addressing for any operations.

Turning next to **Figure 2**, a functional block diagram of a host processor node is depicted in accordance with a preferred embodiment of the present invention. Host processor node **200** is an example of a host processor node, such as host processor node **102** in **Figure 1**. In this example, host processor node **200** shown in **Figure 2** includes a set of consumers **202-208**, which are processes executing on host processor node **200**. Host processor node **200** also includes channel adapter **210** and channel adapter **212**. Channel adapter **210** contains ports **214** and **216** while channel adapter **212** contains ports **218** and **220**. Each port connects to a link. The ports can connect to one SAN subnet or multiple SAN subnets, such as SAN **100** in **Figure 1**. In these examples, the channel adapters take the form of host channel adapters.

Consumers **202-208** transfer messages to the SAN via the verbs interface **222** and message and data service **224**. A verbs interface is essentially an abstract description of the functionality of a host channel adapter. An operating system may expose some or all of the verb functionality through its programming interface. Basically, this interface defines the behavior of the host. Additionally, host processor node **200** includes a message and data service **224**, which is a higher-level interface than the verb layer and is used to process messages and data received through channel adapter **210** and channel adapter **212**. Message and data service **224**

Docket No. AUS920010493US1

provides an interface to consumers **202-208** to process messages and other data.

With reference now to **Figure 3A**, a diagram of a host channel adapter is depicted in accordance with a preferred embodiment of the present invention. Host channel adapter **300A** shown in **Figure 3A** includes a set of queue pairs (QPs) **302A-310A**, which are used to transfer messages to the host channel adapter ports **312A-316A**. Buffering of data to host channel adapter ports **312A-316A** is channeled through virtual lanes (VL) **318A-334A** where each VL has its own flow control. Subnet manager configures channel adapters with the local addresses for each physical port, i.e., the port's LID. Subnet manager agent (SMA) **336A** is the entity that communicates with the subnet manager for the purpose of configuring the channel adapter. Memory translation and protection (MTP) **338A** is a mechanism that translates virtual addresses to physical addresses and validates access rights. Direct memory access (DMA) **340A** provides for direct memory access operations using memory **340A** with respect to queue pairs **302A-310A**.

A single channel adapter, such as the host channel adapter **300A** shown in **Figure 3A**, can support thousands of queue pairs. By contrast, a target channel adapter in an I/O adapter typically supports a much smaller number of queue pairs. Each queue pair consists of a send work queue (SWQ) and a receive work queue. The send work queue is used to send channel and memory semantic messages. The receive work queue receives channel semantic messages. A consumer calls an operating-system specific programming interface, which is herein referred

to as verbs, to place work requests (WRs) onto a work queue.

Figure 3B depicts a switch **300B** in accordance with a preferred embodiment of the present invention. Switch **300B** includes a packet relay **302B** in communication with a number of ports **304B** through virtual lanes such as virtual lane **306B**. Generally, a switch such as switch **300B** can route packets from one port to any other port on the same switch.

Similarly, **Figure 3C** depicts a router **300C** according to a preferred embodiment of the present invention. Router **300C** includes a packet relay **302C** in communication with a number of ports **304C** through virtual lanes such as virtual lane **306C**. Like switch **300B**, router **300C** will generally be able to route packets from one port to any other port on the same router.

Channel adapters, switches, and routers employ multiple virtual lanes within a single physical link. As illustrated in **Figures 3A, 3B, and 3C**, physical ports connect endnodes, switches, and routers to a subnet. Packets injected into the SAN fabric follow one or more virtual lanes from the packet's source to the packet's destination. The virtual lane that is selected is mapped from a service level associated with the packet. At any one time, only one virtual lane makes progress on a given physical link. Virtual lanes provide a technique for applying link level flow control to one virtual lane without affecting the other virtual lanes. When a packet on one virtual lane blocks due to contention, quality of service (QoS), or other considerations, a packet on a different virtual lane is allowed to make progress. Virtual lanes are employed for numerous reasons, some of

Docket No. AUS920010493US1

which are as follows: Virtual lanes provide QoS. In one example embodiment, certain virtual lanes are reserved for high priority or isochronous traffic to provide QoS.

Virtual lanes provide deadlock avoidance. Virtual
5 lanes allow topologies that contain loops to send packets across all physical links and still be assured the loops won't cause back pressure dependencies that might result in deadlock.

Virtual lanes alleviate head-of-line blocking. When
10 a switch has no more credits available for packets that utilize a given virtual lane, packets utilizing a different virtual lane that has sufficient credits are allowed to make forward progress.

With reference now to **Figure 4**, a diagram
15 illustrating processing of work requests is depicted in accordance with a preferred embodiment of the present invention. In **Figure 4**, a receive work queue **400**, send work queue **402**, and completion queue **404** are present for processing requests from and for consumer **406**. These
20 requests from consumer **402** are eventually sent to hardware **408**. In this example, consumer **406** generates work requests **410** and **412** and receives work completion **414**. As shown in **Figure 4**, work requests placed onto a work queue are referred to as work queue elements (WQEs).

25 Send work queue **402** contains work queue elements (WQEs) **422-428**, describing data to be transmitted on the SAN fabric. Receive work queue **400** contains work queue elements (WQEs) **416-420**, describing where to place incoming channel semantic data from the SAN fabric. A
30 work queue element is processed by hardware **408** in the host channel adapter.

The verbs also provide a mechanism for retrieving completed work from completion queue **404**. As shown in **Figure 4**, completion queue **404** contains completion queue elements (CQEs) **430-436**. Completion queue elements

5 contain information about previously completed work queue elements. Completion queue **404** is used to create a single point of completion notification for multiple queue pairs. A completion queue element is a data structure on a completion queue. This element describes a completed

10 work queue element. The completion queue element contains sufficient information to determine the queue pair and specific work queue element that completed. A completion queue context is a block of information that contains pointers to, length, and other information

15 needed to manage the individual completion queues.

Example work requests supported for the send work queue **402** shown in **Figure 4** are as follows. A send work request is a channel semantic operation to push a set of local data segments to the data segments referenced by a

20 remote node's receive work queue element. For example, work queue element **428** contains references to data segment 4 **438**, data segment 5 **440**, and data segment 6 **442**. Each of the send work request's data segments contains a virtually contiguous memory region. The

25 virtual addresses used to reference the local data segments are in the address context of the process that created the local queue pair.

A remote direct memory access (RDMA) read work request provides a memory semantic operation to read a

30 virtually contiguous memory space on a remote node. A memory space can either be a portion of a memory region or portion of a memory window. A memory region

Docket No. AUS920010493US1

references a previously registered set of virtually contiguous memory addresses defined by a virtual address and length. A memory window references a set of virtually contiguous memory addresses that have been
5 bound to a previously registered region.

The RDMA Read work request reads a virtually contiguous memory space on a remote endnode and writes the data to a virtually contiguous local memory space. Similar to the send work request, virtual addresses used
10 by the RDMA Read work queue element to reference the local data segments are in the address context of the process that created the local queue pair. For example, work queue element **416** in receive work queue **400** references data segment 1 **444**, data segment 2 **446**, and
15 data segment **448**. The remote virtual addresses are in the address context of the process owning the remote queue pair targeted by the RDMA Read work queue element.

A RDMA Write work queue element provides a memory semantic operation to write a virtually contiguous memory
20 space on a remote node. The RDMA Write work queue element contains a scatter list of local virtually contiguous memory spaces and the virtual address of the remote memory space into which the local memory spaces are written.

25 A RDMA FetchOp work queue element provides a memory semantic operation to perform an atomic operation on a remote word. The RDMA FetchOp work queue element is a combined RDMA Read, Modify, and RDMA Write operation. The RDMA FetchOp work queue element can support several
30 read-modify-write operations, such as Compare and Swap if equal.

TOP SECRET FRODO

Docket No. AUS920010493US1

A bind (unbind) remote access key (R_Key) work queue element provides a command to the host channel adapter hardware to modify (destroy) a memory window by associating (disassociating) the memory window to a memory region. The R_Key is part of each RDMA access and is used to validate that the remote process has permitted access to the buffer.

In one embodiment, receive work queue **400** shown in **Figure 4** only supports one type of work queue element, which is referred to as a receive work queue element. The receive work queue element provides a channel semantic operation describing a local memory space into which incoming send messages are written. The receive work queue element includes a scatter list describing several virtually contiguous memory spaces. An incoming send message is written to these memory spaces. The virtual addresses are in the address context of the process that created the local queue pair.

For interprocessor communications, a user-mode software process transfers data through queue pairs directly from where the buffer resides in memory. In one embodiment, the transfer through the queue pairs bypasses the operating system and consumes few host instruction cycles. Queue pairs permit zero processor-copy data transfer with no operating system kernel involvement. The zero processor-copy data transfer provides for efficient support of high-bandwidth and low-latency communication.

When a queue pair is created, the queue pair is set to provide a selected type of transport service. In one embodiment, a distributed computer system implementing

Docket No. AUS920010493US1

the present invention supports four types of transport services: reliable, unreliable, reliable datagram, and unreliable datagram connection service.

Reliable and Unreliable connected services associate
5 a local queue pair with one and only one remote queue pair. Connected services require a process to create a queue pair for each process that is to communicate with over the SAN fabric. Thus, if each of N host processor nodes contain P processes, and all P processes on each
10 node wish to communicate with all the processes on all the other nodes, each host processor node requires $P^2 \times (N - 1)$ queue pairs. Moreover, a process can connect a queue pair to another queue pair on the same host channel adapter.

15 A portion of a distributed computer system employing a reliable connection service to communicate between distributed processes is illustrated generally in **Figure 5**. The distributed computer system **500** in **Figure 5** includes a host processor node 1, a host processor node
20 2, and a host processor node 3. Host processor node 1 includes a process A **510**. Host processor node 2 includes a process C **520** and a process D **530**. Host processor node 3 includes a process E **540**.

Host processor node 1 includes queue pairs 4, 6 and
25 7, each having a send work queue and receive work queue.

Host processor node 2 has a queue pair 9 and host processor node 3 has queue pairs 2 and 5. The reliable connection service of distributed computer system **500** associates a local queue pair with one an only one remote
30 queue pair. Thus, the queue pair 4 is used to

Docket No. AUS920010493US1

communicate with queue pair 2; queue pair 7 is used to communicate with queue pair 5; and queue pair 6 is used to communicate with queue pair 9.

5 A WQE placed on one queue pair in a reliable connection service causes data to be written into the receive memory space referenced by a Receive WQE of the connected queue pair. RDMA operations operate on the address space of the connected queue pair.

10 In one embodiment of the present invention, the reliable connection service is made reliable because hardware maintains sequence numbers and acknowledges all packet transfers. A combination of hardware and SAN driver software retries any failed communications. The process client of the queue pair obtains reliable
15 communications even in the presence of bit errors, receive underruns, and network congestion. If alternative paths exist in the SAN fabric, reliable communications can be maintained even in the presence of failures of fabric switches, links, or channel adapter
20 ports.

In addition, acknowledgments may be employed to deliver data reliably across the SAN fabric. The acknowledgment may, or may not, be a process level acknowledgment, i.e. an acknowledgment that validates
25 that a receiving process has consumed the data. Alternatively, the acknowledgment may be one that only indicates that the data has reached its destination.

Reliable datagram service associates a local end-to-end context (EEC) with one and only one remote
30 end-to-end context. The reliable datagram service permits a client process of one queue pair to communicate with any other queue pair on any other remote node. At a

Docket No. AUS920010493US1

receive work queue, the reliable datagram service permits incoming messages from any send work queue on any other remote node.

The reliable datagram service greatly improves scalability because the reliable datagram service is connectionless. Therefore, an endnode with a fixed number of queue pairs can communicate with far more processes and endnodes with a reliable datagram service than with a reliable connection transport service. For example, if each of N host processor nodes contain P processes, and all P processes on each node wish to communicate with all the processes on all the other nodes, the reliable connection service requires $P^2 \times (N - 1)$ queue pairs on each node. By comparison, the connectionless reliable datagram service only requires P queue pairs + (N - 1) EE contexts on each node for exactly the same communications.

A portion of a distributed computer system employing a reliable datagram service to communicate between distributed processes is illustrated in **Figure 6**. The distributed computer system **600** in **Figure 6** includes a host processor node 1, a host processor node 2, and a host processor node 3. Host processor node 1 includes a process A **610** having a queue pair 4. Host processor node 2 has a process C **620** having a queue pair 24 and a process D **630** having a queue pair 25. Host processor node 3 has a process E **640** having a queue pair 14.

In the reliable datagram service implemented in the distributed computer system **600**, the queue pairs are coupled in what is referred to as a connectionless transport service. For example, a reliable datagram service couples queue pair 4 to queue pairs 24, 25 and

Docket No. AUS920010493US1

14. Specifically, a reliable datagram service allows queue pair 4's send work queue to reliably transfer messages to receive work queues in queue pairs 24, 25 and 14. Similarly, the send queues of queue pairs 24, 25, and 14 can reliably transfer messages to the receive work queue in queue pair 4.

In one embodiment of the present invention, the reliable datagram service employs sequence numbers and acknowledgments associated with each message frame to ensure the same degree of reliability as the reliable connection service. End-to-end (EE) contexts maintain end-to-end specific state to keep track of sequence numbers, acknowledgments, and time-out values. The end-to-end state held in the EE contexts is shared by all the connectionless queue pairs communication between a pair of endnodes. Each endnode requires at least one EE context for every endnode it wishes to communicate with in the reliable datagram service (e.g., a given endnode requires at least N EE contexts to be able to have reliable datagram service with N other endnodes).

The unreliable datagram service is connectionless. The unreliable datagram service is employed by management applications to discover and integrate new switches, routers, and endnodes into a given distributed computer system. The unreliable datagram service does not provide the reliability guarantees of the reliable connection service and the reliable datagram service. The unreliable datagram service accordingly operates with less state information maintained at each endnode.

Turning next to **Figure 7**, an illustration of a data packet is depicted in accordance with a preferred embodiment of the present invention. A data packet is a

Docket No. AUS920010493US1

unit of information that is routed through the SAN fabric.' The data packet is an endnode-to-endnode construct, and is thus created and consumed by endnodes. For packets destined to a channel adapter (either host or target), the data packets are neither generated nor consumed by the switches and routers in the SAN fabric. Instead for data packets that are destined to a channel adapter, switches and routers simply move request packets or acknowledgment packets closer to the ultimate destination, modifying the variant link header fields in the process. Routers, also modify the packet's network header when the packet crosses a subnet boundary. In traversing a subnet, a single packet stays on a single service level.

Message data **700** contains data segment 1 **702**, data segment 2 **704**, and data segment 3 **706**, which are similar to the data segments illustrated in **Figure 4**. In this example, these data segments form a packet **708**, which is placed into packet payload **710** within data packet **712**. Additionally, data packet **712** contains CRC **714**, which is used for error checking. Additionally, routing header **716** and transport **718** are present in data packet **712**. Routing header **716** is used to identify source and destination ports for data packet **712**. Transport header **718** in this example specifies the destination queue pair for data packet **712**. Additionally, transport header **718** also provides information such as the operation code, packet sequence number, and partition for data packet **712**.

The operating code identifies whether the packet is the first, last, intermediate, or only packet of a message. The operation code also specifies whether the

Docket No. AUS920010493US1

operation is a send RDMA write, read, or atomic. The packet sequence number is initialized when communication is established and increments each time a queue pair creates a new packet. Ports of an endnode may be
5 configured to be members of one or more possibly overlapping sets called partitions.

In **Figure 8**, a portion of a distributed computer system is depicted to illustrate an example request and acknowledgment transaction. The distributed computer
10 system in **Figure 8** includes a host processor node **802** and a host processor node **804**. Host processor node **802** includes a host channel adapter **806**. Host processor node **804** includes a host channel adapter **808**. The distributed computer system in **Figure 8** includes a SAN fabric **810**,
15 which includes a switch **812** and a switch **814**. The SAN fabric includes a link coupling host channel adapter **806** to switch **812**; a link coupling switch **812** to switch **814**; and a link coupling host channel adapter **808** to switch **814**.

20 In the example transactions, host processor node **802** includes a client process A. Host processor node **804** includes a client process B. Client process A interacts with host channel adapter hardware **806** through queue pair **824**. Client process B interacts with hardware channel
25 adapter hardware **808** through queue pair **828**. Queue pairs **824** and **828** are data structures that include a send work queue and a receive work queue. Process A initiates a message request by posting work queue elements to the send queue of queue pair **824**. Such a work queue element
30 is illustrated in **Figure 4**. The message request of client process A is referenced by a gather list contained

Docket No. AUS920010493US1

in the send work queue element. Each data segment in the gather list points to a virtually contiguous local memory region, which contains a part of the message, such as indicated by data segments 1, 2, and 3, which

5 respectively hold message parts 1, 2, and 3, in **Figure 4**.

Hardware in host channel adapter **806** reads the work queue element and segments the message stored in virtual contiguous buffers into data packets, such as the data packet illustrated in **Figure 7**. Data packets are routed
10 through the SAN fabric, and for reliable transfer services, are acknowledged by the final destination endnode. If not successively acknowledged, the data packet is retransmitted by the source endnode. Data packets are generated by source endnodes and consumed by
15 destination endnodes.

In reference to **Figure 9**, a diagram illustrating the network addressing used in a distributed networking system is depicted in accordance with the present invention. A host name provides a logical identification
20 for a host node, such as a host processor node or I/O adapter node. The host name identifies the endpoint for messages such that messages are destined for processes residing on an end node specified by the host name. Thus, there is one host name per node, but a node can
25 have multiple CAs. A single IEEE assigned 64-bit identifier (EUI-64) **902** is assigned to each component. A component can be a switch, router, or CA.

One or more globally unique ID (GUID) identifiers **904** are assigned per CA port **906**. Multiple GUIDs (a.k.a.
30 IP addresses) can be used for several reasons, some of which are illustrated by the following examples. In one embodiment, different IP addresses identify different

FOIA b 5 - DECLASSIFIED

Docket No. AUS920010493US1

partitions or services on an end node. In a different embodiment, different IP addresses are used to specify different Quality of Service (QoS) attributes. In yet another embodiment, different IP addresses identify different paths through intra-subnet routes.

One GUID **908** is assigned to a switch **910**.

A local ID (LID) refers to a short address ID used to identify a CA port within a single subnet. In one example embodiment, a subnet has up to 2^{16} end nodes, switches, and routers, and the LID is accordingly 16 bits. A source LID (SLID) and a destination LID (DLID) are the source and destination LIDs used in a local network header. A single CA port **1006** has up to 2^{LMC} LIDs **912** assigned to it. The LMC represents the LID Mask Control field in the CA. A mask is a pattern of bits used to accept or reject bit patterns in another set of data.

Multiple LIDs can be used for several reasons some of which are provided by the following examples. In one embodiment, different LIDs identify different partitions or services in an end node. In another embodiment, different LIDs are used to specify different QoS attributes. In yet a further embodiment, different LIDs specify different paths through the subnet. A single switch port **914** has one LID **916** associated with it.

A one-to-one correspondence does not necessarily exist between LIDs and GUIDs, because a CA can have more or less LIDs than GUIDs for each port. For CAs with redundant ports and redundant connectivity to multiple SAN fabrics, the CAs can, but are not required to, use the same LID and GUID on each of its ports.

A portion of a distributed computer system in accordance with a preferred embodiment of the present

2025 RELEASE UNDER E.O. 14176

Docket No. AUS920010493US1

invention is illustrated in **Figure 10**. Distributed computer system **1000** includes a subnet **1002** and a subnet **1004**. Subnet **1002** includes host processor nodes **1006**, **1008**, and **1010**. Subnet **1004** includes host processor nodes **1012** and **1014**. Subnet **1002** includes switches **1016** and **1018**. Subnet **1004** includes switches **1020** and **1022**.

Routers connect subnets. For example, subnet **1002** is connected to subnet **1004** with routers **1024** and **1026**. In one example embodiment, a subnet has up to 216 endnodes, switches, and routers.

A subnet is defined as a group of endnodes and cascaded switches that is managed as a single unit. Typically, a subnet occupies a single geographic or functional area. For example, a single computer system in one room could be defined as a subnet. In one embodiment, the switches in a subnet can perform very fast wormhole or cut-through routing for messages.

A switch within a subnet examines the DLID that is unique within the subnet to permit the switch to quickly and efficiently route incoming message packets. In one embodiment, the switch is a relatively simple circuit, and is typically implemented as a single integrated circuit. A subnet can have hundreds to thousands of endnodes formed by cascaded switches.

As illustrated in **Figure 10**, for expansion to much larger systems, subnets are connected with routers, such as routers **1024** and **1026**. The router interprets the IP destination ID (e.g., IPv6 destination ID) and routes the IP-like packet.

An example embodiment of a switch is illustrated generally in **Figure 3B**. Each I/O path on a switch or

Docket No. AUS920010493US1

router has a port. Generally, a switch can route packets from one port to any other port on the same switch.

Within a subnet, such as subnet **1002** or subnet **1004**, a path from a source port to a destination port is
5 determined by the LID of the destination host channel adapter port. Between subnets, a path is determined by the IP address (e.g., IPv6 address) of the destination host channel adapter port and by the LID address of the router port which will be used to reach the destination's
10 subnet.

In one embodiment, the paths used by the request packet and the request packet's corresponding positive acknowledgment (ACK) or negative acknowledgment (NAK) frame are not required to be symmetric. In one
15 embodiment employing certain routing, switches select an output port based on the DLID. In one embodiment, a switch uses one set of routing decision criteria for all its input ports. In one example embodiment, the routing decision criteria are contained in one routing table. In
20 an alternative embodiment, a switch employs a separate set of criteria for each input port.

A data transaction in the distributed computer system of the present invention is typically composed of several hardware and software steps. A client process
25 data transport service can be a user-mode or a kernel-mode process. The client process accesses host channel adapter hardware through one or more queue pairs, such as the queue pairs illustrated in **Figures 3A, 5, and 6**. The client process calls an operating-system specific
30 programming interface, which is herein referred to as "verbs." The software code implementing verbs posts a work queue element to the given queue pair work queue.

Docket No. AUS920010493US1

There are many possible methods of posting a work queue element and there are many possible work queue element formats, which allow for various cost/performance design points, but which do not affect interoperability.

5 A user process, however, must communicate to verbs in a well-defined manner, and the format and protocols of data transmitted across the SAN fabric must be sufficiently specified to allow devices to interoperate in a heterogeneous vendor environment.

10 In one embodiment, channel adapter hardware detects work queue element postings and accesses the work queue element. In this embodiment, the channel adapter hardware translates and validates the work queue element's virtual addresses and accesses the data.

15 An outgoing message is split into one or more data packets. In one embodiment, the channel adapter hardware adds a transport header and a network header to each packet. The transport header includes sequence numbers and other transport information. The network header
20 includes routing information, such as the destination IP address and other network routing information. The link header contains the Destination Local Identifier (DLID) or other local routing information. The appropriate link header is always added to the packet. The appropriate
25 global network header is added to a given packet if the destination endnode resides on a remote subnet.

If a reliable transport service is employed, when a request data packet reaches its destination endnode, acknowledgment data packets are used by the destination
30 endnode to let the request data packet sender know the request data packet was validated and accepted at the destination. Acknowledgment data packets acknowledge one

"099495-08401"
T09T80"562E660

Docket No. AUS920010493US1

or more valid and accepted request data packets. The requester can have multiple outstanding request data packets before it receives any acknowledgments. In one embodiment, the number of multiple outstanding messages, i.e. Request data packets, is determined when a queue pair is created.

One embodiment of a layered architecture **1100** for implementing the present invention is generally illustrated in diagram form in **Figure 11**. The layered architecture diagram of **Figure 11** shows the various layers of data communication paths, and organization of data and control information passed between layers.

Host channel adapter endnode protocol layers (employed by endnode **1111**, for instance) include an upper level protocol **1102** defined by consumer **1103**, a transport layer **1104**; a network layer **1106**, a link layer **1108**, and a physical layer **1110**. Switch layers (employed by switch **1113**, for instance) include link layer **1108** and physical layer **1110**. Router layers (employed by router **1115**, for instance) include network layer **1106**, link layer **1108**, and physical layer **1110**.

Layered architecture **1100** generally follows an outline of a classical communication stack. With respect to the protocol layers of end node **1111**, for example, upper layer protocol **1102** employs verbs (**1112**) to create messages at transport layer **1104**. Transport layer **1104** passes messages (**1114**) to network layer **1106**. Network layer **1106** routes packets between network subnets (**1116**). Link layer **1108** routes packets within a network subnet (**1118**). Physical layer **1110** sends bits or groups of bits to the physical layers of other devices. Each of the

Docket No. AUS920010493US1

layers is unaware of how the upper or lower layers perform their functionality.

Consumers **1103** and **1105** represent applications or processes that employ the other layers for communicating between endnodes. Transport layer **1104** provides end-to-end message movement. In one embodiment, the transport layer provides four types of transport services as described above which are reliable connection service; reliable datagram service; unreliable datagram service; and raw datagram service. Network layer **1106** performs packet routing through a subnet or multiple subnets to destination endnodes. Link layer **1108** performs flow-controlled, error checked, and prioritized packet delivery across links.

Physical layer **1110** performs technology-dependent bit transmission. Bits or groups of bits are passed between physical layers via links **1122**, **1124**, and **1126**. Links can be implemented with printed circuit copper traces, copper cable, optical cable, or with other suitable links.

The present invention operates within the SAN environment described above with regard to **Figures 1-11**. The present invention provides a mechanism for virtualizing the queue pair space so that the impact of Time-Wait is minimized.

Figure 12 illustrates the process employed by a communication manager to establish a connection between queue pairs (QPs) or end-to-end contexts (EECs) on two different nodes. Typically, the communication manager resides in the host processor node initiating the connection. However, the communication manager may reside in any other node and the present invention is not

Docket No. AUS920010493US1

limited to any particular location of the communication manager.

As shown in **Figure 12**, the communication management request (REQ) message is used to request the
5 establishment of the connection. This message is sent on a well-known QP (such as QP1) that is monitored by management agents on all nodes, i.e. applications on the nodes used to monitor queue pairs. If the management agent residing on the receiving node wishes to accept the
10 request for the connection establishment, the management agent indicates this by responding with a communication management reply (REP) message. The management agent decides whether to accept a request or not based on a number of considerations such as whether it supports the
15 service type requested in the REQ packet, whether it supports the transport service type requested, and whether it has resources, such as a QP, available for the connection.

When the requesting communication manager receives
20 the REP message, the communication manager indicates its receipt and that communication may begin, by sending a ready-to-use (RTU) message. The REQ and REP messages contain information that identifies the type of connection requested and, in particular, the QP number
25 (and EEC number for reliable datagram service) that the sending node wishes to use for this connection.

Figure 13 illustrates the process employed by a communication manager to release a connection between QPs or EECs on two different nodes. The communication
30 management disconnect request (DREQ) message is used to request the release of the connection. When the management agent residing on the receiving node receives

Docket No. AUS920010493US1

the DREQ, the management agent indicates that the connection may be released by responding with a communication management disconnect reply (DREP) message. The DREQ message contains an identifier that uniquely
5 identifies the connection to be released, and the QP number or EEC number of the remote node that is associated with this connection.

When the responding node receives the DREQ, the responding node places the QP or EEC associated with this
10 connection into a time-wait state. During the time-wait state, the QP or EEC may not be used for another connection, so essentially it remains idle. The time-wait state is used to ensure that all data packets being transmitted along the original connection are
15 routed to their respective end nodes.

When the initiating node receives the DREP, the initiating node places its QP or EEC associated with this connection into the time-wait state. The QP or EEC remains in the time-wait state for a sufficient time to
20 allow any packets or acknowledgments to traverse the SAN fabric, so that after the time-wait period has elapsed, there will be no more packets received for this connection. If the QP was reused immediately, a packet may be received on the QP from the old connection (this
25 is quite possible as there may be packets in flight that were sent before the DREQ was received). This packet may be misinterpreted as being related to the new connection and may cause data integrity or security exposures to the application.

30 **Figure 14** illustrates two nodes that have a connection established between two QPs. Node 1 **1410** that is using the virtualization process of the present

Docket No. AUS920010493US1

invention, has a large pool of virtualized QPs **1450** from which one of the virtualized QPs **1452** may be allocated to the connection between node 1 **1410** and node 2 **1420**. The pool of virtualized QPs **1450** is initially allocated at
5 initialization time when the HCA and its physical QPs are initialized. There is a bit associated with each virtualized QP that indicates whether it is available or in the time-wait state. This bit and the QP number itself are the only resources associated with the
10 virtualized QP. This pool **1450** may be large and may have any number of virtual QPs. For example, the pool **1450** may use up to the full 24 bit number space for queue pairs, as it does not consume any real resources in the channel adapter (CA). The physical QP space is much
15 smaller as it requires hardware resources in the CA that are used for the transmission and reception of packets.

When a virtualized QP, such as virtual QP **1452**, is allocated to a connection, that QP number is associated with the physical QP **1460** that is used for the
20 connection. The particular QP allocated to a connection is selected using a selection scheme. The selection scheme may be any known selection scheme, such as simply allocating the next QP in the stack that is not being used and is not in a time-wait state, using a random
25 selection scheme, or the like. The communication manager requesting the connection typically performs the allocation of the QP.

In the depicted example, Node 2 **1420** is not using the virtualization process of the present invention and
30 thus, only associates a physical QP number with the connection. While **Figure 14** shows only Node 1 **1410** using the virtualized QPs of the present invention, the

1099495-00404
1099495-00404

Docket No. AUS920010493US1

invention is not limited to such an exemplary embodiment. Rather, both Nodes 1 and 2 may make use of the present invention. In addition, there may be any number of nodes upon which the present invention may be implemented and
 5 the present invention is not limited to only connections between two nodes.

As part of the virtualization process, Node 1 **1410** allocates QP2 **1452** from the larger pool of virtualized QPs **1450** and associates the virtual queue pair QP2 **1452**
 10 with the physical queue pair QP **1460**. The difference between the virtual queue pair QP2 **1452** and the physical queue pair QP **1460** is that the physical QP is where the WQEs are stored and the virtualized QP only consumes a QP number and a bit to indicate whether the virtual QP is
 15 available.

In all communication management messages associated with this connection the communication manager, which may be in either one of node 1 or node 2, identifies Node 1's QP as virtual queue pair QP 2 **1452**, not physical queue
 20 pair QP **1460**. As depicted in **Figure 14**, Node 1 **1410** has virtual queue pair QP 2 **1452** connected to physical queue pair QP 4 **1470** on Node 2 **1410**. From Node 2's perspective, it has physical queue pair QP4 **1470** connected to virtual queue pair QP2 **1452** on Node 1, and
 25 knows nothing about the existence of physical queue pair QP **1460**.

When the connection between virtual queue pair QP2 **1452** and physical queue pair QP4 **1470** is no longer needed, the communication manager requests that the
 30 connection be disconnected using the DREQ and DREP protocol described earlier. The virtual queue pair QP2

T09T00"50T000

Docket No. AUS920010493US1

1452 and physical queue pair QP4 **1470** are both placed in the time-wait state, and cannot be used again until the time-wait period has expired.

Now, suppose, as illustrated in **Figure 15**, Process A
5 on Node 1 **1510** needs to establish a connection with Process D on Node 2 **1520**. Node 2 **1520** cannot re-use the physical queue pair QP4 **1570** for this connection until the time-wait period has expired. Therefore, Node 2 **1520** must use another physical queue pair, if there is one
10 available.

As shown in **Figure 15**, the physical queue pair QP5 **1580** is allocated to the new connection. Similarly, Node 1 **1510** cannot re-use virtual queue pair QP2 **1552** until the time-wait period has expired. However, QP2 **1552** is a
15 virtualized QP and is not consuming CA hardware resources. Thus, another virtual queue pair QP from the virtualized pool **1550** may be assigned to the new connection, e.g., virtual queue pair QP3 **1554**.

The physical queue pair QP **1560** may be reused for
20 this new connection because it has not been placed in a time-wait state. The physical queue pair QP **1560** is then associated with the virtual queue pair QP 3 **1554** for this new connection. Thus, a new connection is established that connects virtual queue pair QP3 **1554** on Node1 with
25 physical queue pair QP 5 **1580** on Node 2 **1520**. In both the new and the old connection the physical queue pair QP **1560** is used by Node 1 **1510**. Therefore, this optimizes the use of the hardware resources on the CA of Node 1 **1510**.

Figure 16 is a flowchart outlining an exemplary operation of the present invention. As shown in **Figure 16**, the operation starts with receiving a request to establish a new connection (step **1610**). If this request is from another node that is attempting to initiate a connection, the request may be a REQ message.

A next virtual queue pair from the virtual queue pair pool is selected for the new connection (step **1620**). A determination is made as to whether this virtual queue pair is in a time-wait state or is already allocated to a connection (step **1630**). If so, the operation returns to step **1620** and the next virtual queue pair in the pool is allocated.

If the virtual queue pair is not in a time-wait state and is not already allocated to a connection, the virtual queue pair is allocated to the new connection (step **1640**). A message is then sent to the other node with which a communication connection is to be established (step **1650**). If the present node is the initiator, this message may be a REQ message. If the present node is a receiver of a REQ message from an initiating node, the message may be a REP message.

Thereafter, assuming that if the present node is the initiator a RTU message is received from the other node, communication between the two nodes may begin (step **1660**). The operation then ends.

The preceding description illustrates how virtualized QPs can be used to optimize the use of hardware resources, when the transport type is reliable connected or unreliable connected, a similar technique can be used to virtualize end-to-end contexts and thus, save hardware resources associated with an end-to-end

Docket No. AUS920010493US1

context when using the Reliable Datagram transport type.

Figure 17 illustrates the manner by which the present invention may be applied to end-to-end contexts (EECs). As shown in **Figure 17**, similar to the virtual queue pairs in the embodiments described above, a pool of virtual EECs is provided which may be allocated to connections between node 1 and node 2. These virtual EECs receive communications via the physical QP of the node and the physical EECs, as shown. In the same way that the virtual QPs are placed in a time-wait state when a connection is torn down between node 1 and node 2, the virtual EECs are placed in a time-wait state when the connection between nodes 1 and 2 is torn down. This enables the physical EECs to be immediately reallocated to another connection using a new virtual EEC, e.g., EEC2 in **Figure 17**.

Thus, the present invention provides a virtualized queue pair pool that may be used in managing connections between physical queue pairs. By use of the virtualized queue pairs of the present invention, the problems associated with processes waiting for a time-wait period to expire are avoided. That is, without this invention, new connection requests may need to be rejected until one or more time-wait periods have elapsed to make the hardware resources available. In large fabrics the time-wait period may be quite long. However, with the present invention, the hardware resources are made available immediately by virtue of the virtualized queue pairs being placed in the time-wait state.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary

Docket No. AUS920010493US1

skill in the art will appreciate that the processes of
the present invention are capable of being distributed in
the form of a computer readable medium of instructions
and a variety of forms and that the present invention
5 applies equally regardless of the particular type of
signal bearing media actually used to carry out the
distribution. Examples of computer readable media
include recordable-type media such as a floppy disk, a
hard disk drive, a RAM, and CD-ROMs and transmission-type
10 media such as digital and analog communications links.

The description of the present invention has been
presented for purposes of illustration and description,
but is not intended to be exhaustive or limited to the
invention in the form disclosed. Many modifications and
15 variations will be apparent to those of ordinary skill in
the art. The embodiment was chosen and described in
order to best explain the principles of the invention,
the practical application, and to enable others of
ordinary skill in the art to understand the invention for
20 various embodiments with various modifications as are
suited to the particular use contemplated.

TE9T50" 663T550